SFLaaS: Software Fault Localization as a Service

Qusay Idrees Sarhan[1, 2], Hassan Bapeer Hassan[3], and Árpád Beszédes[1]
[1] Department of Software Engineering, University of Szeged, Szeged, Hungary
[2] Department of Computer Science, University of Duhok, Duhok, Iraq
[3] Department of Medicine, University of Duhok, Duhok, Iraq
{sarhan, beszedes}@inf.u-szeged.hu, hassan.bapeer@uod.ac

***Abstract*—Many tools for enabling developers to locate bugs in their programs have been proposed in the literature. The majority of programs they target are based on C/C++ and Java. In this paper, we offer a tool named "SFLaaS" for locating faults in programs written in Python and is provided as a service rather than as a plugin or a command-line tool to be installed. Thus, our tool can be accessed anytime and from anywhere. The tool employs Spectrum-based fault localization (SBFL) to help Python developers automatically analyze their programs and generate useful data at run-time to be used to produce a ranked list of potentially faulty program elements (i.e., statements). Our tool supports different important features in fault localization such as supporting about 80 SBFL formulas, different tie-breaking methods, showing code elements with different colors, ranging from most suspicious (red) not suspicious (green) based on their suspicious scores, allowing the user to define his/her own formula, etc. Using our tool could help developers to efficiently find the locations of different types of faults in their programs.**

I. Introduction

Programs play an important role in our day-to-day activities. Nonetheless, errors and faults still exist in most of them. Some of them are critical that may have serious consequences. Thus, several software fault localization approaches have been implemented, such as Spectrum-based fault localization (SBFL) [1]. In SBFL, the level of suspiciousness from being faulty for each program entity is computed depending on the program spectra acquired by performing a set of test cases. However, it is not widespread in the industry sector as it has some issues. One problem is that most of the SBFL tools focus on C/C++ and Java programs [2]. Therefore, it lacks the support for developers to debug their software using other languages such as Python.

In this paper, we implement a tool named "SFLaaS" as a service to enable the software fault localization process, which can be used anywhere and anytime. This tool is useful for Python developers to easily analyze their software by generating data to produce a list of suspicious elements at runtime. To mark an element as suspicious, the element should be examined by the developer from the top of the list to the bottom (from most suspicious element to least one). To examine the applicability of this tool, a Python program can be uploaded to see the results. The outcome shows that our tool is easy to use and beneficial in finding faults in Python programs.

II. Background of SBFL

To obtain the spectra of the targeted program, test case executions on the program elements are stored at the beginning of the SBFL process. This way, a two-dimensional spectrum is generated that demonstrates the connection between program elements and its test cases. Program elements and test cases are presented by their rows and columns respectively. A matrix cell demonstrates if the related program element (row) is covered by the related test case (column). Also, the matrix contains an extra row for the test results (passed or failed).

Then, for each element e, four statistical numbers could be computed: (a) ep: number of passed test cases covering e; (b) ef: number of failed test cases covering e; (c) np: number of passed test cases not covering e; (d) nf: number of failed test cases not covering e. Then, these four basic statistics can be used by an SBFL formula [3] such as Tarantula, Ochiai, or Barinel to compute the suspicion score of each element.

Eventually, the output will be generated as a ranking list based on the scores. The highest element in the ranking list is the most suspicious to contain a fault. Therefore, it is easier for the developers to discover faults in a target program.

III. Fault Localization as a Service

A. SFLaaS's Architecture

We run the test cases on the target program using "pytest"[1] to fetch the results. To collect the program's spectra on the statement level, code coverage measurement is required. The program must be instrumented in order to generate the code coverage. Therefore, the popular Python coverage measuring framework, called "coverage.py" [2] has been used in our tool.

Next, the tool constructs coverage and test results from the gathered data [4], and finally, based on the specified SBFL formulas, it scores the suspiciousness of each element.

B. SFLaaS's User Interface

The user interface of SFLaaS is shown in Figure 1. It can be noted that many options are provided for the user to start the software fault localization process.

The main features of SFLaaS are listed below:

1) Accessibility: not like plugin, command-line, or standalone tools; our tool can be accessed anytime and from anywhere as it is provided as a service. Thus, the user only needs a browser and an internet connection.

[1]https://docs.pytest.org/en/7.1.x/
[2]https://coverage.readthedocs.io/en/6.4.2/

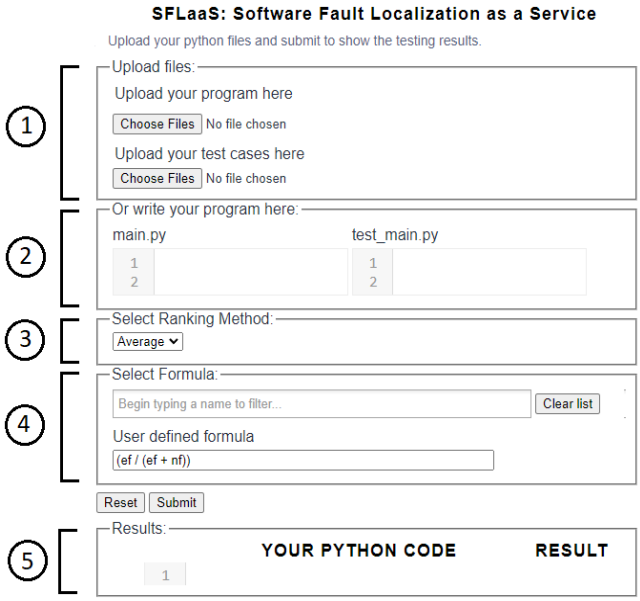

Fig. 1. Main user interface of SFLaaS

2) Easy upgrades: our tool does not require manual installation, configuration, or updates on the user's side as the service provider deals with hardware and software updates; thus removing this workload and responsibility from the user.

3) Code uploading/editing: The user has two options to submit their program and its tests to the tool: (a) the user uploads a python program file and its related tests file using buttons made for this purpose, as shown in part (1) of Figure 1, and (b) the user writes their program and its tests in a specific editing area specified for this purpose, as shown in part (2) of Figure 1. This is useful especially when the tool is used for educational purposes such as teaching students fault localization.

4) Tie-breaking methods: this option, part (3) of Figure 1, enables the user to select a tie-breaking method (e.g., Min, Max, or Average) and apply it to the elements sharing the same suspicion score in the list.

5) Formulas selection: this option, part (4) of Figure 1, enables the user to select one or more SBFL formulas (e.g., Tarantula, Ochiai, Barinel, etc.). In our tool, we have implemented about 80 formulas that have been proposed in the literature. This is especially important for researchers who would like to compare the efficiency of different SBFL formulas with each other. Also, it enables the user to define his/her own formula either by combining existing formulas or by introducing new formulas via combining different statistical numbers (i.e., ef, ep, nf, np). This is crucial when comparing newly proposed formulas to the existing ones.

6) Highlighted code elements: when the SBFL is performed, the results will be shown in part (5) of Figure 1; where the corresponding code elements will be highlighted with different colors, ranging from red (most suspicious) to green (not suspicious), based on the scores as shown in Figure 2.

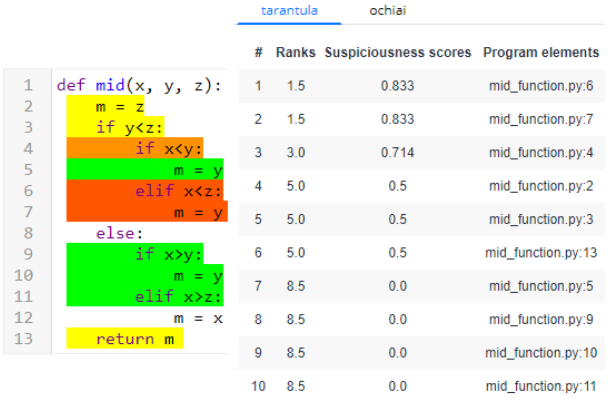

Fig. 2. Highlighted statements based on suspiciousness scores

7) Navigation: the SBFL results in Figure 2 present the program elements with their positions in the source code, ranks, and scores. Clicking on an element in the SBFL results table puts the cursor at the element's location in the source code in order to be easily examined by the user.

C. How to use SFLaaS

In this section, we will describe how our tool can be used to locate faults in Python programs. The user starts the fault localization process by clicking on the "Submit" button, in Figure 1, and then looks at the produced ranking list of suspicious statements of the uploaded program. The user clicks on the first statement in the list with the highest score, the tool redirects the user to the statement location in the source code file and tries to see if it is a bug or not. If it is a bug, then the user has to fix it. Then, the user re-runs the test cases and notices the pass of all the test cases. This indicates that the bug is fixed. It is worth mentioning that each uploaded program gets deleted after its execution on the server side; this is very important to ensure data protection. Only the top ten elements from the ranking list are explored by the developers because after that, they begin to lose the desire to follow up the fault localization tools [5], [6]. Thus, any fault localization tool could be considered successful, if the most faulty elements are listed in the top-10 ranks. In this situation, excessive amounts of energy and time could be conserved by using our tool.

IV. Conclusions

This paper describes "SFLaaS"[3], a fault localization tool for Python programs that is provided in form of software as a service. It is implemented with many helpful and practical characteristics to aid developers in debugging their programs. Various use cases have been shown to assess the relevance of this tool. It locates the existing faults in various programs easily and in a straightforward way. For future work, we plan to add interactivity for developers to comment on the results, thus enhancing them via re-ranking. Also, supporting various techniques to visualize the results. Finally, a user study to evaluate tool usability and acceptance will be conducted.

[3]https://sflaas.daxazi.com/

REFERENCES

[1] C. Gouveia, J. Campos, and R. Abreu, "Using html5 visualizations in software fault localization," in *First IEEE Working Conference on Software Visualization (VISSOFT)*, 2013, pp. 1–10.

[2] Q. I. Sarhan and A. Beszedes, "A survey of challenges in spectrum-based software fault localization," *IEEE Access*, vol. 10, pp. 10 618–10 639, 2022.

[3] Neelofar, "Spectrum-based Fault Localization Using Machine Learning," 2017. [Online]. Available: https://findanexpert.unimelb.edu.au/scholarlywork/1475533-spectrum-based-fault-localization-using-machine-learning

[4] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, 2002, pp. 467–477.

[5] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 165–176. [Online]. Available: https://doi.org/10.1145/2931037.2931051

[6] X. Xia, L. Bao, D. Lo, and S. Li, "Automated debugging considered harmful: A user study revisiting the usefulness of spectra-based fault localization techniques with professionals using real bugs from large systems," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 267–278.